Libwallaby Tello Support

This note is a first stab at documenting the tello support for libwallaby. I probably have missed several things, so feel free to ask and I will update this note.

Note: this modification to libwalley has Camera::Device defaulting to the TELLO – this can be changed by commenting out camera.cpp line: 292 where BOTUI_TELLO_TEST is defined. This is done to facilitate testing with BOTUI and must be changed before production.

Packages needed or updated

Rng-tools – random number seed, greatly improves wifi speed

Ffmpeg update

Built with these parameters, etc
ffmpeg version N-99349-g184fc42 Copyright (c) 2000-2020 the FFmpeg developers
built with gcc 4.9.2 (Raspbian 4.9.2-10)

Use the download from ffmpeg.org and when running configure used the following configuration:

./configure --arch=armel --target-os=linux --enable-gpl --enable-omx --enable-omx-rpi --enable-nonfree --enable-mmal --enable-shared

libavutil 56. 59.100 / 56. 59.100
libavcodec 58.106.100 / 58.106.100
libavformat 58. 58.100 / 58. 58.100
libavdevice 58. 11.102 / 58. 11.102
libavfilter 7. 87.100 / 7. 87.100
libswscale 5. 8.100 / 5. 8.100
libswresample 3. 8.100 / 3. 8.100
libpostproc 55. 8.100 / 55. 8.100

Opencv update (v3 or branch 3.4) – Had to rebuild because the "stock" opencv and ffmeg were specifically linked together

Use the download from opencv.org git branch: 3.4

When building with cmake, use the following parameters:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
   -D CMAKE_INSTALL_PREFIX=/usr/local \
   -D INSTALL_C_EXAMPLES=OFF \
   -D INSTALL_PYTHON_EXAMPLES=OFF \
   -D OPENCV_GENERATE_PKGCONFIG=ON \
   -D ENABLE_NEON=ON \
   -D OPENCV_EXTRA_EXE_LINKER_FLAGS=-latomic \
   -D ENABLE_VFPV3=ON \
   -D BUILD_TESTS=OFF \
   -D OPENCV_ENABLE_NONFREE=ON \
   -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules \
   -D BUILD_EXAMPLES=OFF ..
```

Included in libwallaby/src

UDPvideo.cpp

UDPvideo primarily consists of two threads, one thread to take UDP packets from the tello and create video frames and the second thread converts those frames to a format that libwallaby can use. Both threads and the queues between them are designed to ensure that the user software receives the most recent frame from the tello.

run_VDR – receives UDP packets from the tello. Uses av_parser_parse2 to reconstruct the original frame. Every time a frame is ready it is passed to run_VFP. Note: every time an I-frame is detected, the frame queue between run_VDP and run_VFP is flushed to ensure only the most recent frames are forwarded and to minimize CPU cycles.

run_VFP – decodes frames to a format that libwallaby expects and forwards it to wait for the user process to consume it. Note: Only the most recent frame will be forwarded to ensure that the most recent frame is consumed.

tello.cpp

wpa_cmd - used to send a command to wpa_supplicant

tellos_find – returns a list of tellos in a string

tello _connect– connects to a single tello

tello_send – sends a string (or command) to a tello

wpa_sup_connect – sets up a connection to wpa_supplicant

wpa_ctrl (directory)

A set of support functions hacked from wpa_client to support tello.cpp.

/etc/network/interfaces

```
# interfaces(5) file used by ifup(8) and ifdown(8)
# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'
# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d
auto lo
iface lo inet loopback
allow-hotplug wlan0
iface wlan0 inet dhcp
```

/home/pi/wifi_configurator.py

```
#
# Wesley Myers
# wmyers@kipr.org
#
# This script is designed to configure Wi-Fi for the Wallaby.

import os
import hashlib
import time

# ===== generate the hostapd.conf file =====

#generate the ssid
f = os.popen('wallaby_get_id.sh')
wallaby_id = f.read()

f = os.popen('wallaby_get_serial.sh')
wallaby_serial = f.read()

#gives unique password for the AP
hash_id=hashlib.sha256(wallaby_id).hexdigest()[0:6]+'00'

# figure out which channel is being used by the client

f = os.popen( 'iwlist wlan0 channel | grep \"Current Frequency:\" | awk -F \'(\' \'{gsub("\)", "",
$2); print $2}\' | awk -F \' \' \'{print $2}\' ' )
wlanChannel = f.read()

best_channel = wlanChannel

print "The best channel is " + str(best_channel)
best_channel=str(best_channel)

#strings for the /etc/hostapd_wallaby.conf file
interface='interface=uap0\n'
country='country_code=US\n'
ssid='ssid=' + wallaby_serial + '-wombat\n'
channel='channel=' + best_channel + '\n'
wpa='wpa=3\n'
wpa_passphrase='wpa_passphrase=' + hash_id + '\n'

os.system('chmod +x /etc/hostapd_wallaby.conf')
```

```
#remove any old files
os.system('rm /etc/hostapd_wallaby.conf')

print "Setting up hostapd_wallaby.conf"
f=open('/etc/hostapd_wallaby.conf', 'w')
f.write(interface)
f.write(country)
f.write(ssid)
f.write(channel)
f.write(wpa)
f.write(wpa_passphrase)
f.close()

# ===== Start Wi-Fi =====

# ==== Virtual Interfaces ====

os.system("hostapdstart")
print "hostapdstart done"
time.sleep(10)

os.system('wall -n \"Wombat Name: ' + wallaby_serial + '-wombat\nPassword: ' + hash_id + '\"')

#alert user via sound
os.system('/usr/bin/amixer set PCM 100%')
```

===end==code===

/usr/local/bin/hostapdstart

```
echo "**deleting uap0"
iw dev uap0 del
echo "**creating uap0"
iw dev wlan0 interface add uap0 type __ap

#Modify iptables (these can probably be saved using iptables-persistent if desired)
echo "IPV4 forwarding: setting..."
#sysctl net.ipv4.ip_forward=1
#echo "net.ipv4.ip_forward=1" >> /etc/sysctl.conf
sed -i 's/^#net.ipv4.ip_forward=.*$/net.ipv4.ip_forward=1/' /etc/sysctl.conf
echo "Editing IP tables..."
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -F
iptables -t nat -F
sleep 2
iptables -t nat -A POSTROUTING -s 192.168.125.0/24 ! -d 192.168.125.0/24 -j MASQUERADE
iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
iptables -A FORWARD -i wlan0 -o uap0 -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -i uap0 -o wlan0 -j ACCEPT
#iptables-save > /etc/iptables/rules.v4
iptables-save > /etc/iptables.ipv4.nat
#iptables-restore < /etc/iptables.ipv4.nat

echo "**bringing uap0 up"
ifconfig uap0 192.168.125.1 netmask 255.255.255.0 broadcast 192.168.125.1
ifup --verbose uap0
echo "**starting hostapd"
sleep 10
hostapd -dd -t -B /etc/hostapd_wallaby.conf
sleep 20
echo "**restarting dnsmasq service"
service dnsmasq restart
echo "**dnsmasq started"
```

/etc/dhcpcf.conf

```
# A sample configuration for dhcpcd.
# See dhcpcd.conf(5) for details.

# Allow users of this group to interact with dhcpcd via the control socket.
#controlgroup wheel

# Inform the DHCP server of our hostname for DDNS.
hostname

# Use the hardware address of the interface for the Client ID.
clientid
# or
# Use the same DUID + IAID as set in DHCPv6 for DHCPv4 ClientID as per RFC4361.
#duid

# Persist interface configuration when dhcpcd exits.
persistent

# Rapid commit support.
# Safe to enable by default because it requires the equivalent option set
# on the server to actually work.
option rapid_commit

# A list of options to request from the DHCP server.
option domain_name_servers, domain_name, domain_search, host_name
option classless_static_routes
# Most distributions have NTP support.
option ntp_servers
# Respect the network MTU.
# Some interface drivers reset when changing the MTU so disabled by default.
```

```
#option interface_mtu

# A ServerID is required by RFC2131.
require dhcp_server_identifier

# Generate Stable Private IPv6 Addresses instead of hardware based ones
slaac private

# A hook script is provided to lookup the hostname if not set by the DHCP
# server, but it should not be run by default.
nohook lookup-hostname
#denyinterfaces uap0
interface uap0
   static ip_address=192.168.125.1
   nohook wpa_supplicant
```

/etc/dnsmasq.conf

```
interface=lo,uap0
no-dhcp-interface=lo,wlan0
listen-address=192.168.125.1
#bind-interfaces
server=8.8.8.8
#domain-needed
bogus-priv
dhcp-range=192.168.125.100,192.168.125.200,24
```

Network setup for Access Point (AP) + Client on the same hardware.

Important! If the AP & Client is to be used, then the client must be started first and connected its endpoint. The AP then must be started using the same channel as the Client. Otherwise, the wifi will become unusable and most likely (as I have not figured out how to recover) will require a reboot to recover. Note: the hardware capabilities indicate that the wifi does support 2 channels, but it seems that the driver does not…

There are three files (two scripts and one configuration) used in the test platform:

/etc/network/interfaces – will cause the client interface to be brought up at startup time. A connection on this interface must be made before the AP is brought up.

/home/pi/wifi_configurator.py **–** Modified from the original script at the same location. Basic function is to create the configuration file (/etc/hostapd_wallaby.conf) used by hostapd. Important functions include, creating the ssid, etc and finding the client interface channel number.

Normally, this script is started by zoobee_launcher.sh, which is a startup script that also starts botui. However, for a Tello enabled Wombat *wifi_configurator.py* will need to started differently.

/usr/local/bin/hostapdstart – configures the ap0 interfaces and kicksoff hostapd.

Various Tello Configurations

Three basic configurations:

(1) Video Tello
(2) Video Tello + Tello swarm
(3) Tello Swarm

Running a video tello

Running a video tello requires that the access point (AP) not be active. This is because there is no way to know which channel the tello will choose and having the AP active forces the wifi to be on a single channel and my limited testing seems show having the AP active prevents a connection through the client.

With only the client active, all three of the tello_test programs provide a method to connect and send commands to the tello as well as processing video.

Running a video Tello with a swarm of Tellos

So to run a video Tello (needs client) along with a swarm of Tellos this has to happen:

(1) Configure the Tellos in the swarm to be in station mode and to connect to the wallaby AP when the Tello boots. (tello_test_swarm gives an example of how to do this for a single Tello.) This will require the wallaby to be in client mode and the AP being off.
(2) (untested) Make sure that the swarm Tellos are powered off. It might be okay tp let them reboot, but it seems they will persistently try to connect to the wallaby's AP. So, I am being conservative here.
(3) Connect the Wombat to the video Tello. Let it start the video stream. (tello_test_camera gives an example of this)
(4) Bring up the access point. During this process, the video stream will appear to stop and will after a couple of second start, but should keep flowing.
(5) Startup the swarm Tellos, they should start connecting to the AP. I do not have an example of managing a connecting swarm as I wanted to get this out. However, there are several examples (in c and python) on how to do this. If I get a chance, I'll try to create a library function to manage swarms for the programmer.

Running a Tello swarm

(1) Configure the Tellos in the swarm to be in station mode and to connect to the wallaby AP when the Tello boots. (tello_test_swarm gives an example of how to do this for a single Tello.) This will require the wallaby to be in client mode and the AP being off.
(2) Bring up the access point
(3) Startup the swarm Tellos, they should start connecting to the AP. I do not have an example of managing a connecting swarm as I wanted to get this out. However, there are several examples (in c and python) on how to do this. If I get a chance, I'll try to create a library function to manage swarms for the programmer.

Note: In both the swarm examples the Tellos will stay in station mode and attempt to connect to the Wombat that they are configured to until they are reset. So, step 1 should be necessary only once.

Libwallaby test programs

tello_swarm_test – simply places the Tello into station mode. The Tello will then reboot and attempt to connect to the Wombat

Issue - need to add wallaby hostname/pw, currently this is hard coded

tello_botui_test - Tells the Tello to stream out video and then loops.

tello_camera_test - same as *camera_speed_test*, except find a Tello, start the video stream and then read and output blob info.

Tello video testing with Botui

At this time of pull request there is not menu-driven means to configure a Tello as the camera input for Botui. This can be done by having a Tello send video packets to the Wallaby and configuring the camera functions to default to TELLO instead of BLACK_2017. That way when Botui wants camera input the camera function will automatically start receiving the Tello video.

The process is a follows:

(1) Make sure that BOTUI_TELLO_TEST is defined in camera.cpp. The define is in line 10.
(2) Rebuild libwallaby and install.
(3) Restart Botui. Botui should link with the rebuilt libwallaby
(4) Make sure a Tello is active and in AP mode.
(5) Run tests/tello_botui_test. This should find the active tello and get it to start transmitting video.
(6) In botui, “motors and sensors”->Camera. You should be able to see what the Tello is transmitting.

Issues and bugs

Plenty of debug statements (Wombat seems to handle well)

Deprecated code for ffmpeg that needs to be attended to